AR-001-179

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

ELECTRONICS RESEARCH LABORATORY

TECHNICAL REPORT

ERL-0004-TR

POLYNOMIAL MANIPULATION WITH APL

B. Billard

S U M M A R Y

A simple but effective system for the manipulation of polynomials of several variables in APL is presented. The system is especially advantageous in situations where more sophisticated symbolic computing systems are not available, or have failed to solve particular problems. The system is shown to successfully solve a problem not resolved by a more sophisticated system.

POSTAL ADDRESS: Chief Superintendent, Electronics Research Laboratory, Box 2151, G.P.O., Adelaide, South Australia, 5001.

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

1. INTRODUCTION

A quarter century of development in the area of symbolic computing has resulted in a wide ranging heritage of symbolic computing systems. Excellent "state of the art" reviews of this area have resulted from the 1966 and 1971 Symposia on Symbolic and Algebraic Manipulation(ref.1,2). Considering the unquestioned power of these systems, it is perhaps surprising that they have not gained a wider acceptance within the scientific community as a standard tool. Part of the problem may be the slow process of education as to what is available. The categorisation by Moses(ref.3) of systems into "conservative", "liberal", "new left", and "catholic" eloquently highlights another problem - that of deciding what the user wants, and indeed what can be reasonably provided. Systems which attempt to be all things to all people may become so complex that their larger computing requirements (time, memory, user education and systems support) discourage their incorporation within many computing environments.

There is therefore a requirement for a smaller scale, more "simple minded" formalism which will allow isolated researchers to harness some of the power of symbolic computing even though they may not have access to (or may not know about) the more extensive systems such as FORMAC, ALTRAN, REDUCE, SCHOONSCHIP, and SIN.

This paper presents techniques by which manipulation of polynomials of an arbitrary number of variables may be carried out using APL. A minimum of two simple APL routines are required (for polynomial addition and multiplication), but the simplicity of the formalism is such that it is easy to extend the range of functions to allow for, say, polynomial division, and the manipulation of rational polynomials.

APL has not been fully exploited as a medium for symbolic computing, although its potential is undoubted. The use of APL for manipulation of polynomials of a single variable has long been recognised, and it is now included in basic texts(ref.4). Other users(ref.5,6,7) have pointed to the possibilities, while generally limiting their attention to specific areas. More detailed comments will be given later on Surkan's application(ref.7).

APL is especially suited to symbolic computing for several reasons. Firstly, the programmer does not have to worry about storage control. Routines may be coded with complete generality, without regard to the rank or size of arrays. The storage problems caused by "intermediate expression swell" in some other systems (see Tobey(ref.8)) and "garbage collection" are therefore not apparent to the user. Secondly, APL functions can be used as operators which, for our purposes, allow polynomial expressions to be built up naturally and executed without the use of an intermediate interpreter. Further, APL is designed to be used interactively. This is a decided advantage when the form of a problem solution is not known in advance. The user is in a position to massage the output of the symbolic computation in a way which he determines at that time.

The approach and the routines presented in this paper were developed to solve a particular problem in filtering theory. A solution had been attempted using FORMAC, but failed because an unknown intermediate term went outside the allowable range. This raises a further advantage of this simpler approach.

The system presented here is sufficiently transparent so that if something does go wrong in the computations, the user is in a good position to work out how he can circumvent the problem - either by reformulating it, or by writing his own routines to handle it. The presentation here is therefore deliberately open-ended, to point the reader towards the possibilities rather than to spell out all the details.

The paper assumes a knowledge of APL.

2. POLYNOMIAL REPRESENTATION

Consider a polynomial in the r variables $x_1,\ldots,x_r$. The polynomial $a_{i_1,\ldots,i_r} \; x_1^{i_1},\ldots,x_r^{i_r}$ may be represented as an array of rank r

$$A = (a_{i_1,\ldots,i_r}) \tag{1}$$

Thus the index of each term in A is 1 greater than the power to which each variable x_i is raised. Alternatively, if the APL index origin is set to zero ($\square$ IO←0), the term $A[i_1;\ldots;i_r]$ gives the coefficient of the term in $x_1^{i_1} \ldots x_r^{i_r}$. This representation is in general more expansive than is strictly necessary, since a large proportion of the elements of A may be zero. A more compact form would be one such as that adopted by ALTRAN, which, for N non zero terms, would represent A by only N.(r+1) items(ref.9). This more compact form was also adopted by Surkan(ref.7) in his APL implementation. However our simplicity of representation means that coding of routines to manipulate the arrays is much simpler.

The polynomial form (1) has been used for the basic arithmetic operations add, multiply and divide between polynomials, but it is capable of extension to allow manipulation of rational polynomials, using as building blocks the routines to handle the form (1).

One possible representation is to place the numerator and denominator "back to back". Thus the numerator is represented by the array

$$A[0;;\ldots.], \tag{2}$$

and the denominator by

$$A[1;;\ldots.].$$

Alternatively, the denominator may be kept as a series of factors, so that the numerator is represented by

$$A[0;;\ldots.], \tag{3}$$

as in (2), but

$$A[i;;\ldots], \quad i > 0$$

are factors of the denominator.

The form (3) would be less efficient with storage, since the arrays $A[i;;\ldots]$ would all have to be filled out to the same shape. However, in problems where the same factors occur frequently in denominators, this may be less demanding than searching to eliminate common factors from the form (2).

3. POLYNOMIAL MULTIPLICATION

If P_A and P_B are polynomials represented by the arrays of coefficients A and B respectively, then the polynomial product $P_A P_B$ can be found by forming the outer product A∘.×B, and summing those elements of the outer product which correspond to the same exponents in the polynomial product. The procedure for selecting and summing these elements can be most easily visualised as a generalization of the single variable polynomial product.

If A and B represent polynomials of degree n and m respectively, in x, the outer product will be an (n+1)x(m+1) array of products $(a_i b_j)$, $i = 0,\ldots,n$, $j = 0,\ldots m$, (where $A = (a_0,\ldots,a_n)$ and $B = (b_0,\ldots,b_m)$). The coefficient of x^k in the polynomial product will be the sum of all terms $a_i b_j$ such that $i+j=k$. Thus we must sum all elements in the outer product along the reverse diagonal $i+j=k$ for each $k = 0,1,\ldots,n+m$ (see figure 1).

There are two ways in which the elements along each reverse diagonal of the outer product may be selected. The first is to generate a selector matrix which has the same shape as the outer product, and which is zero everywhere except for 1's along the appropriate reverse diagonal. The APL product of this selector matrix with the outer product will, when reduced by summation, give the desired coefficient of a term in the polynomial product. This technique has been implemented in APL and has been found to be adequate, but slower than would be desirable. The slowness is due to the need to separately shape the selector matrix for each term in the polynomial product.

The second technique which has been implemented is performed by rotating the outer product matrix so that the diagonals $i+j=k$ become columns, which may then be reduced by summation in a single APL operation.

The process is illustrated for the single variable case in figure 2, where the outer product has been padded with n columns of zeros to the left, and the rows then rotated by amounts ranging from n for the first row, to 0 for the last row. Reduction by summation over the first axis then gives a vector which is the array representation of the polynomial product.

For the generalization to polynomials of r variables, assume P_A and P_B are of degree $n_1,\ldots,n_r$ and $m_1,\ldots,m_r$ in each of the variables respectively. The outer product A∘.×B will have shape $(n_1+1),\ldots,(n_r+1),(m_1+1),\ldots,(m_r+1)$, and the product will be of degree $p_i = n_i+m_i, i=1,\ldots,r$. If we pad the outer product in the same way as for the single variable case, the resulting array will have shape

$$(n_1+1),\ldots,(n_r+1),(p_1+1),\ldots,(p_r+1),$$

that is, we have padded n layers of zeros onto the left side of the (r+i)th axis for each i.

A separate rotation will now need to be carried out for each variable. For the first variable, this means defining an array C of shape

$$(n_1+1),\ldots,(n_r+1),(p_2+1),\ldots,(p_r+1)$$

such that

C[0;...;] is n,

C[1;...;] is n -1,

.....

and

C[n ;...;] is 0.

After the rotation has been performed, each section of the array perpendicular to the (r+1)st axis will be characterised by (i_1+j_1) = constant, where i_1 is the index of the first and j_1 is the index of the (r+1)st axis respectively. Each such section would have the same shape as C.

If this rotation operation is now carried out with respect to the second variable in the same way, the section of shape

$$(n_1+1),\ldots,(n_r+1),(p_3+1),\ldots,(p_r+1)$$

which is perpendicular to the (r+1)st and (r+2)nd axes will be characterised by i_1+j_1 = constant and i_2+j_2 = constant.

After similar rotations for each of the r variables, each section perpendicular to the (r+1)st,...,(2r)th axes will have shape $(n_1+1),\ldots,(n_r+1)$, and will be characterised by

$$i_s+j_s = \text{constant} = k_s, \quad s = 1,\ldots,r$$

Thus elements contributing to each term in the product are located in columns defined by the first r axes of the array, and the array representing the polynomial product may be formed by reduction by summation over each of these first r axes.

While this technique is a little more difficult to visualize than that involving the generation of a selector matrix, it is shorter (30 times shorter for multiplication of polynomials of degree 9 in two variables) because the bulk of the operations are carried out once for each axis (variable), rather than once for each term in the product.

4. POLYNOMIAL DIVISION

The system used for polynomial division, while it thoroughly exploits APL techniques and therefore may look quite different, is identical in concept to that outlined by Collins(ref.10) for the PM system.

Consider two polynomials A and B in r variables. We assume A is of degree n_i in x_i and B is of degree m_i in x_i. If $n_i < m_i$ for any i we know B will not divide A and hence without loss of generality we may assume $n_i \geq m_i, i=1,\ldots,r$.

We start by defining the most significant term. This is done by defining a primary ordering of the non-zero terms according to the exponent of x_1, a secondary ordering according to the exponent of x_2 and so on. In APL this is implemented by recursively removing outer layers of zeros, until the array is at its smallest significant size, and then dropping all except the last layer of the remnant array in the first dimension on the first cycle, the second dimension on the second cycle, and so on to the r-th dimension on the r-th cycle.

The most significant term in B can then be divided into the most significant term in A; the result, q say, added to the quotient Q, and a new A'=A-Bq formed in the usual manner of polynomial long division. The process will halt when the leading term in B no longer divides the leading term in A'. A' will then be an array which has only zero elements for $(i_1 > m_1, i_2 > m_2, \ldots, i_r > m_r)$. Note that this does not necessarily mean that the degree of Q in, say x_s, is less than m_s. In fact Q may still be of degree n_s in each of the variables $x_1,\ldots,x_r$.

(E.g. consider $A = x^2+y^2+xy$ and $B = xy$). In 2 dimensions this may be illustrated by representing the remainder A' as

$$A' = \begin{pmatrix} A_0 & A_2 \\ A_1 & 0 \end{pmatrix}$$

where A_0 has shape n,m and A_1 and A_2 are not necessarily zero.

5. FURTHER CONSIDERATIONS

The basic arithmetic operations, multiplication, division and addition are sufficient to solve a wide range of polynomial manipulation problems. Listings of APL routines to perform these functions are given in Appendix I.

The most expensive of the basic routines ADD, BY, and DIV in terms of storage requirements is the polynomial multiplication routine, BY. This routine must accommodate an array equal in size to the outer product of one factor and the polynomial product. Under normal circumstances, and especially in a virtual storage environment, this should not cause problems. However the applicability of the routine can be extended if a checkpoint is added before the formation of the outer product in BY, at which the available working area, □WA, is tested against requirements. If necessary one of the factors can then be split (presumably half way along its longest axis) for a recursive call to BY. If this occurs then the section of BY which processes outer products will be bypassed in the primary call. Because lowest powered exponents are eliminated in BY before formation of the outer product, this procedure may considerably reduce the workload if either of the factors is large and has widely spaced non-zero elements. The listing of BY in Appendix I includes such a checkpoint with a rudimentary test of available space. Tests carried out on the self-multiplication of a 10 x 10 non-zero array showed that the execution time was increased by about 50% when recursion was carried to the fourth level.

There are further operations which may usefully be performed by using the basic routines as building blocks.

One desirable facility is to be able to eliminate common factors from two polynomials, say, numerator and denominator. A traditional approach to this has been the search for a greatest common divisor. In theory this can be achieved by a straightforward application of Euclid's theorem, but in practice it has proved to be one of the great problem areas of symbolic computing (see, for example, Collins(ref.11)).

A simpler but more limited approach that has been used successfully by the author has been to build a "library" of factors which arise naturally in denominators in the course of a series of calculations, and to look for factors from amongst these. If a series of routines (say ADDR, BYR and OVER) are built up from the basic ADD and BY to handle rational polynomial functions, then it would be appropriate in OVER to check to see if a new factor could be added to the library of denominator factors.

Another useful facility is the ability to substitute new expressions for variables. A routine, REDRNK, has been written to perform this and its listing is included in the appendix. The only limitation is that the user may need to transpose co-ordinates of the input polynomial so that substitution is made for the correct variable (the last one). If the substituted polynomial includes a new variable the input polynomial will also need to be recast so that it is a function (albeit trivial) of that variable. Both these operations are straightforward in APL.

Further functions such as polynomial differentiation can also be easily coded in APL.

6. AN APPLICATION

The following application is one which arose in the course of research in the area of Kalman filtering and smoothing. The aim was to determine the bandwidths of a particular Kalman filter-smoother as a function of the bandwidths of the Kalman filter without the smoother.

Under steady state conditions the filtering and smoothing matrices may be described in terms of 3 parameters f_s, f_v and f_a, which are proportional to the filter's initial response to a step in position, velocity and acceleration respectively. In fact, f_s will approximate the (non-dimensional) frequency response of the filter with respect to position. (The same cannot be said of f_v and f_a because of phase shifts.)

Only one of these parameters may be specified independantly, and we may write, for $t = \sqrt{(1-f_s)}$,

$$f_s = (1-t^2) \; ; \; f_v = 2(1-t)^2 \; ; \; f_a = (1-t)^3 / (1+t)$$

If b_s, b_v, b_a are the respective smoothing responses, it is possible to show that they may be defined so that they are the diagonal elements of a 3 x 3 matrix W, given by

$$W = (PS-\Phi)(I-A^{-1})$$

where
$$\Phi = \begin{pmatrix} 1 & 1 & \frac{1}{2} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} a/d & -1 & \frac{1}{2} \\ b/d & 1 & -1 \\ 2c/d & 0 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} f_s, & f_v & , 2f_a \\ f_v, & f_v\,(12-10f_s-f_v)/(8f_s\,(1-f_s)) & , f_a(2f_s-f_v)/(1-f_s) \\ 2f_a, & f_a((2f_s-f_v)/(1-f_s) & , 2f_af_v/(1-f_s) \end{pmatrix}$$

$$a = (1-f_v+f_a); \quad b = (f_v-2f_a); \quad c = f_a; \quad \text{and } d = (1-f_s),$$

and S is the solution of the equation

$$S-A^TSA = \Gamma\,\Phi$$

where
$$\Gamma = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Here P is proportional to the covariance of the filtered state vector, Γ is a scaled information matrix, and Φ the transition matrix for a unit data rate.

The only significant remaining problem is the solution of the equation for S. Since A is algebraically simpler than A, this may be equivalently expressed as

$$(A^{-1})^T\,S(A^{-1}) - S = (A^{-1})^T\,\Gamma\,\Phi\,(A^{-1})$$

It was the solution of this set of 9 equations in the 9 unknowns $S_{ij}, i,j=1,2,3$ which was attempted using the standard procedures of FORMAC and which failed because some intermediate term had gone beyond the allowable magnitude range. Solution of the equations was then attempted by hand. Although a range of relatively simple subrelations could be derived, they were still too cumbersome to solve by hand.

The relations were

$$\begin{aligned} S_{11} &= S_{12} + S_{21} \\ S_{12} &= 2(S_{13} + S_{22}) \\ S_{21} &= 2(S_{31} + S_{22}) \\ 8cS_{33} &= 1 + 2(d-1)\ (S_{13}+S_{31}) \\ S_{31} &= S_{32} = -S_{23} \\ 2(1+d)(S_{13}+S_{31}) &+ 2(2a+b+2d)\ S_{22} = 1 \\ 2(b+c-d(b+3c))\ (S_{13} &- S_{31}) = b+c \\ \text{and}\quad 2(b+c-d(b+3c))\ S_{31} &= -d \end{aligned}$$

Solution was again attempted using FORMAC, which again failed.

The subrelations were then coded in APL using the polynomial manipulation routines ADD and BY and the results were used to calculate the responses b_s, b_v, b_a, as pairs of polynomials (numerator and denominator) in two variables t, and $\theta = (1+t)^{-1}$. The solutions that were obtained are listed in Appendix II.

The validity of the polynomial solutions was confirmed by the numerical solution of the equations for S and W for a range of f_s. However, the primary form of the polynomial solution as numerators WN11, WN22, WN33, and denominators WD11, WD22, WD33, was still too complex to give insight, and it was therefore desirable to simplify these expressions by use of polynomial manipulation.

The first step was to eliminate common factors by testing each numerator and denominator against a library of 9 factors which arose in the course of the calculations. The second step was to make use of the rational function nature of the result to eliminate the variable θ. This was done first by substituting $\zeta = 1/\theta$ (effected by reversing the order of the last co-ordinate in the numerator and denominator) and then by substituting $\zeta = (1+t)$. After testing the results for factors (1+t) and (1-t), it was found that

$$\begin{aligned} b_s &= (1+6t-34t^3+39t^4-4t^5-8t^6)/(t^2(1+t^2)(1+4t+t^2)) \\ b_v &= (1-t^2)/(1+4t+t^2) \\ \text{and}\quad b_a &= 4t(1-t)/((1+t)(1+4t+t^2)) \end{aligned}$$

and it is remembered that $t^2 = 1-f_s$.

The output of the two steps in the simplification is included in Appendix II.

7. DISCUSSION

Some points of comparison should be made between the APL system of polynomial manipulation presented in this paper, and those proposed previously by Surkan(ref.7). In his paper, Surkan reported the development of APL routines to add, subtract, multiply and differentiate polynomials. He used a different form of polynomial representation in which each term is represented by its coefficient and the exponents of the (ordered set of) variables.

Tests were run with Surkan's example of orbit calculation to compare the effectiveness of the two approaches.

The problem essentially amounts to a recursive set of calculations involving polynomials in the variables U, V, and W, given that $F_0 = 1$, $G_0 = 0$,

$$F_N = \dot{U}\frac{\partial F_{N-1}}{\partial U} + \dot{V}\frac{\partial F_{N-1}}{\partial V} + \dot{W}\frac{\partial F_{N-1}}{\partial W} - UG_{N-1}$$

$$G_N = U\frac{\partial G_{N-1}}{\partial U} + V\frac{\partial G_{N-1}}{\partial V} + W\frac{\partial G_{N-1}}{\partial W} + F_{N-1}$$

and $\dot{U} = -3UV$, $\dot{V} = W-2V$, $\dot{W} = -V(U-2W)$. (Note that the equation for $\dot{V}$ was listed incorrectly in Surkan's paper, although it was programmed correctly in his figure 2.)

The current set of routines performed these calculations to N = 19 in 18.2 s on an IBM 370/168 with VSAPL. This compares most favourably with Surkan's quoted time of 20 min on an APL online terminal to an IBM 360/50, in spite of the difference in systems.

Of more consequence is the difference in the results at N = 11. Although some terms agree with those of Surkan's figure 3, most do not. Only the addition routine was listed in Surkan's figure 1, and so it is difficult to be more precise about where the difference arises. Since the same equations programmed in FORMAC give answers identical with those from the current set of routines, it must be assumed either that this author has misinterpreted the example, or else the routines reported in reference 7 were in some way faulty. A listing of the correct solution for N = 11 is given in Appendix III.

It is interesting to note that the FORMAC solution took 5.65 s in primary execution time and 8.43 s including the preprocessing, compilation, and link-editing steps. This does not diminish the value of the APL approach for the reasons discussed in section 1. In fact the ratio of execution times, at a little more than 2:1, appears to be substantially less than the 10 - 100:1 generally accepted for execution times of problems coded in APL to those for coding in other higher level languages (such as FORTRAN and PL/1)(12). The advantage with APL normally comes with the time to program and debug, and this will dominate in programs which do not require repeated execution. It is this 'problem solving' orientation of APL which is especially suited to symbolic computing.

REFERENCES

No.	Author	Title
1	All papers	Comm. ACM 14, August 1971
2	All papers	Comm. ACM 9, August 1966
3	Moses, J.	"Algebraic Simplification : A Guide to the Perplexed". Comm. ACM 14, pp 527 - 537, August 1971
4	Iverson, K.E.	"APL in Exposition". IBM Philadelphia Scientific Centre, Report 320 - 3010
5	Bergquist, J.W.	"Algebraic Manipulation". APL 74, Proceedings, pp 45 - 49
6	Kellerman, A.	"APL Symbolic Manipulation and Generating Functions". APL 75, Proceedings, pp 214 - 220
7	Surkan, A.J.	"Symbolic Polynomial Operations with APL". IBM J. Res. Develop., pp 209 - 211, March 1969
8	Tobey, R.G.	"Experience with FORMAC Algorithm Design". Comm. ACM 9, pp 589 - 597, August 1966
9	Hall, A.D.	"The ALTRAN System for Rational Function Manipulation - A Survey". Comm. ACM 14, pp 517 - 521, August 1971
10	Collins, G.E.	"PM, A System for Polynomial Manipulation" Comm. ACM 9, pp 578 - 589, August 1966
11	Collins, G.E.	"Subresultants and Reduced Polynomial Remainder Sequences". ACM Journal 14, pp 128 - 142, January 1967
12	Streeter, D.N.	"Cost-benefit Evaluation of Scientific Computing Services". IBM Sys. J. 11, pp 219 - 233, 1972

APPENDIX I

APL PROGRAM LISTINGS

```
      ∇BY[⎕]∇
      ∇ RES←A BY B;⎕IO;C;RM;I;J;N;RNK;RA;RB;FAC
[1]   RNK←(⍴⍴A)⌈(⍴⍴B)⌈⎕IO←0
[2]   ⍝ CHECK FOR NULL OR TRIVIAL POLYNOMIAL
[3]   →((0=+/,|A)∨(0=+/,|B))/NULL
[4]   →((0<⌊/⍴B←STRIP B)∧(0<⌊/⍴A←STRIP A))/NZ
[5]   NULL:→0×⍴⍴RES←(RNK⍴1)⍴0
[6]   TA:→0×⍴⍴RES←B×+/,A
[7]   TB:→0×⍴⍴RES←A×+/,B
[8]   NZ:→((1=⌈/⍴A),(1=⌈/⍴B),((⍴⍴A)=⍴⍴B))/(TA,TB,S1)
[9]   B←(((RNK-⍴⍴B)⍴1),⍴B)⍴B
[10]  A←(((RNK-⍴⍴A)⍴1),⍴A)⍴A
[11]  ⍝  REMOVE COMMON POWERS FROM A AND B
[12]  S1:RB←(⍴B)+0×RA←(⍴A)+FAC←RNK⍴I←0
[13]  L0:N←I+1+⍳(RNK-I+1)+J←0
[14]  LA:→(0=+/,|(RA[⍳I],(J←J+1),RA[N])↑A)/LA
[15]  J←0×RA[I]←RA[I]-FAC[I]←J-1
[16]  A←(-RA)↑A
[17]  LB:→(0=+/,|(RB[⍳I],(J←J+1),RB[N])↑B)/LB
[18]  RB[I]←RB[I]-J←J-1
[19]  B←(-RB)↑B
[20]  →(RNK>I←I+1+0×FAC[I]←FAC[I]+J)/L0
[21]  ⍝ CHECK AVAILABLE WORK AREA
[22]  →(0<⎕WA-40+(20×RNK)+×/20,RA,RM←RA+RB-1)/S2
[23]  I←1+0×J←⌈/RA
[24]  L:→(J≠(I-1)↓(I←I+1)↑RA)/L
[25]  RES←(-N+⍴RES)↑RES←B BY(N←((I-1)⍴0),(J←⌈0.5×J),(RNK-I)⍴
      0)↓A
[26]  →(RNK=⍴RM←⍴RES←RES ADD B BY(((I-1)↑RA),J,I↓RA)↑A)/RESET
[27]  S2:I←0×⍴⍴RES←(RA,(-RM))↑A∘.×B
[28]  L1:N←((I↑RA),1,((I+1)↓RA),(I↑RM),(I+1)↓RM)
[29]  →(RA[I]=1↑I↓⍴C←N⍴J←RA[I]-1)/EQ1
[30]  L2:→(RA[I]>1↑I↓⍴C←C,[I]N⍴J←J-1)/L2
[31]  EQ1:→((I←I+1)<0.5×⍴⍴RES←C⌽⌈RNK+I⌉RES)/L1
[32]  L3:→(RNK<⍴⍴RES←+/[0]RES)/L3
[33]  ⍝ RESTORE COMMON POWERS TO RESULT
[34]  REST:RES←STRIP(-RM+FAC)↑RES
      ∇
      ∇ADD[⎕]∇
      ∇ RES←A ADD B;R
[1]   ⍝ ENSURE A AND B HAVE THE SAME RANK
[2]   →((⍴⍴A)=⍴⍴B)/S1
[3]   R←(⍴⍴A)⌈⍴⍴B
[4]   B←(((R-⍴⍴B)⍴1),⍴B)⍴B
[5]   A←(((R-⍴⍴A)⍴1),⍴A)⍴A
[6]   ⍝ PAD OUT SMALLER OF EACH DIMENSION
[7]   S1:RES←STRIP(R↑A)+(R←(⍴A)⌈⍴B)↑B
      ∇
      ∇STRIP[⎕]∇
      ∇ R←STRIP A;RA;RNK;I;J;A1;A2
[1]   ⍝ REDUCES A TO SMALLEST SIGNIF. SIZE
[2]   →((+/,|A)=+/,R←((1⌈RNK←⍴⍴A)⍴1)⍴A)/0
[3]   I←1+J←0×⍴⍴R←A
[4]   L1:A1←(I-1)↑RA←⍴R
[5]   A2←I↓RA
[6]   L2:→(0=+/,|(A1,(-J←J+1),A2)↑R)/L2
[7]   →((I←I+1)≤⍴⍴R←(((I-1)⍴J←0),(1-J),(RNK-I)⍴0)↓R)/L1
      ∇
```

```
      ∇DIV[⎕]∇
      ∇ RES←A DIV B;I;⎕IO;RNK;Q;AS;RS;CB;RCB;RDQ;RSQ
[1]   ⍝ POL.A ÷ POL.B, RETURNS QUOTIENT AND REMAINDER
[2]    ⎕IO←1+0×+/,RNK←(⍴⍴A←STRIP A)⌈⍴⍴B←STRIP B
[3]    →((⍴⍴A)=⍴⍴B)/S0
[4]    A←(((RNK-⍴⍴A)⍴1),⍴A)⍴A
[5]    B←(((RNK-⍴⍴B)⍴1),⍴B)⍴B
[6]   S0:→(1<⌈/⍴B)/S1
[7]    →0×⍴⍴RES←(2,⍴Q)↑(1,⍴Q)⍴Q←(A÷+/,B)
[8]   ⍝ FIND THE HIGHEST POWERED TERM IN B
[9]   S1:RCB←RNK⍴0×I←1
[10]   RS←⍴CB←B
[11]  LB:RCB[I]←RCB[I]+RS[I]
[12]   →((I←I+1)≤⍴RS←⍴CB←STRIP CB←(((I-1)↑RS),¯1,I↓RS)↑CB)/LB
[13]   Q←(RNK⍴1)⍴0
[14]  ⍝ CAN A QUOTIENT BE FOUND?
[15]  L1:→(0>⌊/((⍴A)-⍴B))/NO
[16]   →(0=+/,|AS←(RSQ←(RCB)-I←1)↓A)/NO
[17]   RS←⍴AS←STRIP AS
[18]  L2:RSQ[I]←RSQ[I]+RS[I]
[19]   →((I←I+1)≤⍴RS←⍴AS←STRIP AS←(((I-1)↑RS),¯1,I↓RS)↑AS)/L2
[20]  ⍝ RSQ IS HIGHEST POWER, AND AS ITS COEFFICIENT
[21]   RDQ←RSQ-RCB-1+I←0
[22]   AS←+/,AS÷CB
[23]   Q←Q ADD RDQ⍴(((×/RDQ)-1)⍴0),AS
[24]   →(RNK=⍴⍴A←STRIP A ADD-(1-RDQ+⍴B)↑B×AS)/L1
[25]  NO:RES←(RS↑Q),[0.5](RS←(⍴Q)⌈⍴A)↑A
      ∇
      ∇RATPOL[⎕]∇
      ∇ RES←A RATPOL B;⎕IO;I;RNK;RA;RB;JA;JB;N
[1]    RNK←(⍴⍴A)⌈(⍴⍴B)⌈1
[2]   ⍝ ENSURE A AND B HAVE THE SAME RANK
[3]    →((RNK=⍴⍴A)∧RNK=⍴⍴B)/S1
[4]    B←(((RNK-⍴⍴B)⍴1),⍴B)⍴B
[5]    A←(((RNK-⍴⍴A)⍴1),⍴A)⍴A
[6]   S1:→((0=+/,|A)∨(0=+/,|B))/S2
[7]   ⍝ REMOVE COMMON POWERS OF VARIABLES
[8]    I←⎕IO←0×+/(RA←⍴A),RB←⍴B
[9]   L1:N←I+1+⍳(RNK-I+1)↑JA←JB←0
[10]  LA:→(0=+/,|(RA[⍳I],(JA←JA+1),RA[N])↑A)/LA
[11]  LB:→(0=+/,|(RB[⍳I],(JB←JB+1),RB[N])↑B)/LB
[12]   RA[I]←RA[I]-JA←(JA⌊JB)-1
[13]   RB[I]←RB[I]-JA
[14]   A←(-RA)↑A
[15]   B←(-RB)↑B
[16]   →(RNK≠I←I+1)/L1
[17]  S2:RES←(2,RA)⍴(,RA↑A),,(RA←(⍴A)⌈⍴B)↑B
      ∇
      ∇REDRNK[⎕]∇
      ∇ RES←A REDRNK B;⎕IO;RN;RB
[1]   ⍝ SUBSTITUTES POL.,A, FOR THE LAST VARIABLE OF B
[2]    →(1=¯1↑(⍴B)+0×RN←⍴RB←(-⎕IO←1)↓⍴RES←B)/OUT
[3]   L1:→(1<¯1↑⍴RES←(((RN⍴0),¯1)↓RES)ADD A BY(RB,(1-¯1↑⍴RES)
      )↑((RB←¯1↓⍴RES),¯1)↑RES)/L1
[4]   OUT:RES←(¯1↓⍴RES)⍴RES
      ∇
```

```
      ∇REDFAC[⎕]∇
      ∇ RES←PLIB REDFAC RPOLY;FAC;⎕IO;TK;N;D;NFACT;IF;P;Q;RRL
[1]   ⍝ ELIMINATES FACTORS IN PLIB FROM RPOLY(RAT. POL.)
[2]    N←TK⍴(1,TK←(1↓⍴RPOLY))↑RPOLY
[3]    D←TK⍴(¯1,TK)↑RPOLY
[4]    NFACT←+/(1↑⍴PLIB)
[5]    RRL←(⍴⍴PLIB)-⎕IO←1+IF←0
[6]   ⍝ SELECT NEXT FACTOR
[7]   NEWF:→(NFACT<IF←IF+1)/NOMORE
[8]    FAC←(1↓⍴FAC)⍴FAC←STRIP((IF-NFACT),RRL⍴0)↓((IF-1),RRL⍴0)↓
       PLIB
[9]   ⍝ IS IT A FACTOR OF NUMERATOR AND DENOMINATOR?
[10]  L:→(0<+/,|(¯1,(1↓⍴Q))↑Q←N DIV FAC)/NEWF
[11]   →(0<+/,|(¯1,(1↓⍴P))↑P←D DIV FAC)/NEWF
[12]   N←TK⍴(1,TK←1↓⍴Q)↑Q
[13]   D←TK⍴(1,TK←1↓⍴P)↑P
[14]   →((⍴⍴FAC)≠+/⍴FAC)/L
[15]  NOMORE:RES←(P↑N),[0.5](P←(⍴N)⌈⍴D)↑D
      ∇
```

APPENDIX II

FILTER PROBLEM SOLUTION

WN11 and WD11 are the numerator and denominator respectively of the first diagonal term of W, expressed as a rational polynomial in t and θ. WT1 is the same term expressed as a rational polynomial in the form (2), after elimination of common factors using the routine REDFAC and the library of factors, PLIB. W1 is the same term after elimination of θ. Similar comments apply to the other diagonal terms of W.

RATPOL takes two polynomials as input and returns them in the rational polynomial form (2), after cancelling factors which are powers of the variables

```
      WN11
 0      0      0      0      0
 0     32    ¯32      0      4
 0   ¯288    288     16    ¯48
 0   1160  ¯1156   ¯144    264
 0  ¯2744   2712    576   ¯880
 0   4200  ¯4088  ¯1344   1980
 0  ¯4312   4088   2016  ¯3168
 0   2968  ¯2688  ¯2016   3696
 0  ¯1320   1096   1344  ¯3168
 0    344   ¯232   ¯576   1980
 0    ¯40      8    144   ¯880
 0      0      4    ¯16    264
 0      0      0      0    ¯48
 0      0      0      0      4
      WD11
 0     0     0
 0     0     0
 0     0     0
 0     8    ¯4
 0   ¯40    24
 0    72   ¯64
 0   ¯40   104
 0   ¯40  ¯120
 0    72   104
 0   ¯40   ¯64
 0     8    24
 0     0    ¯4
      □←WT1←2×PLIB REDFAC WN11 RATPOL WD11
 8   ¯8    0    1
¯24   24    4   ¯6
 26  ¯25  ¯12   15
¯10    8   12  ¯20
  0    1   ¯4   15
  0    0    0   ¯6
  0    0    0    1

  0    0    0    0
  0    0    0    0
  2   ¯1    0    0
  2    0    0    0
  0   ¯1    0    0
  0    0    0    0
  0    0    0    0
      □←W1←(2 2ρ1 1 1 ¯1)REDFAC(1 2 1ρ1 1)REDRNK⌸WT1
 1    6    0  ¯34   39   ¯4   ¯8
 0    0    1    6   10    6    1
      ⍉WN22
 0     0     0    12   ¯56   100   ¯80    20     8    ¯4
 0     0     0    ¯8    32   ¯40     0    40   ¯32     8
      ⍉WD22
 0    0    0    8  ¯16   ¯8   32   ¯8  ¯16    8    0
 0    0    0   ¯4   12  ¯12    4    4  ¯12   12   ¯4
```

```
      □←WT2←PLIB REDFAC WN22 RATPOL WD22
 3  ¯2
¯2   0
¯1   2
 0   0

 2  ¯1
 4  ¯1
 2  ¯1
 0  ¯1
      □←W2←(2 2⍴1 1 1 ¯1)REDFAC(1 2 1⍴1 1)REDRNK⌽WT2
 1   0  ¯1
 1   4   1
      ⌽WN33
 0    0    0    0    0    0    0    0    0    0
 0    0    0    4  ¯12    8    8  ¯12    4    0
 0    0    0   ¯4   24  ¯60   80  ¯60   24   ¯4
      ⌽WD33
 0    0    0    8  ¯16    0   16   ¯8    0
 0    0    0   ¯4   12  ¯16   16  ¯12    4
      □←WT3←.5×PLIB REDFAC WN33 RATPOL WD33
 0   1  ¯1
 0  ¯1   4
 0  ¯1  ¯6
 0   1   4
 0   0  ¯1

 2  ¯1   0
 0   1   0
¯2  ¯1   0
 0   1   0
 0   0   0
      □←W3←(2 2⍴1 1 1 ¯1)REDFAC(1 2 1⍴1 1)REDRNK⌽WT3
 0   4  ¯4   0
 1   5   5   1
```

APPENDIX III

ORBIT PROBLEM SOLUTION
('F11',[1]A)FORM SEL F

$$F11= 1023\,U^5V - 878268\,U^4V^3 + 93660\,U^4VW + 36761454\,U^3V^5$$
$$- 14873940\,U^3V^3W + 1189902\,U^3VW - 318715236\,U^2V^7$$
$$+ 234084492\,U^2V^5W - 53057340\,U^2V^3W^2 + 3479700\,U^2VW^3$$
$$+ 654729075\,UV^9 - 695674980\,UV^7W + 266431410\,UV^5W^2$$
$$- 42723300\,UV^3W^3 + 2320275\,UVW^4$$

$$G11= -U^5 + 53640\,U^4V^2 - 1008\,U^4W - 6379326\,U^3V^4$$
$$+ 1657140\,U^3V^2W - 40446\,U^3W^2 + 97257888\,U^2V^6 - 57948120\,U^2V^4W$$
$$+ 9297840\,U^2V^2W^2 - 255000\,U^2W^3 - 310134825\,UV^8 + 290768940\,UV^6W$$
$$- 93324150\,UV^4W^2 + 11213100\,UV^2W^3 - 308745\,UW^4$$

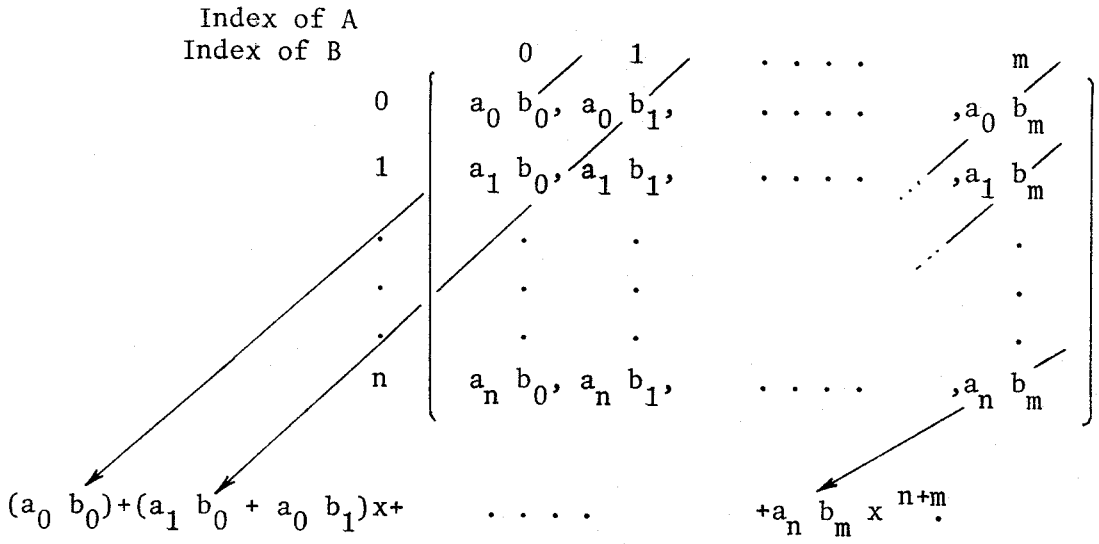

Figure 1. Formation of the product of polynomials of a single variable from the outer product

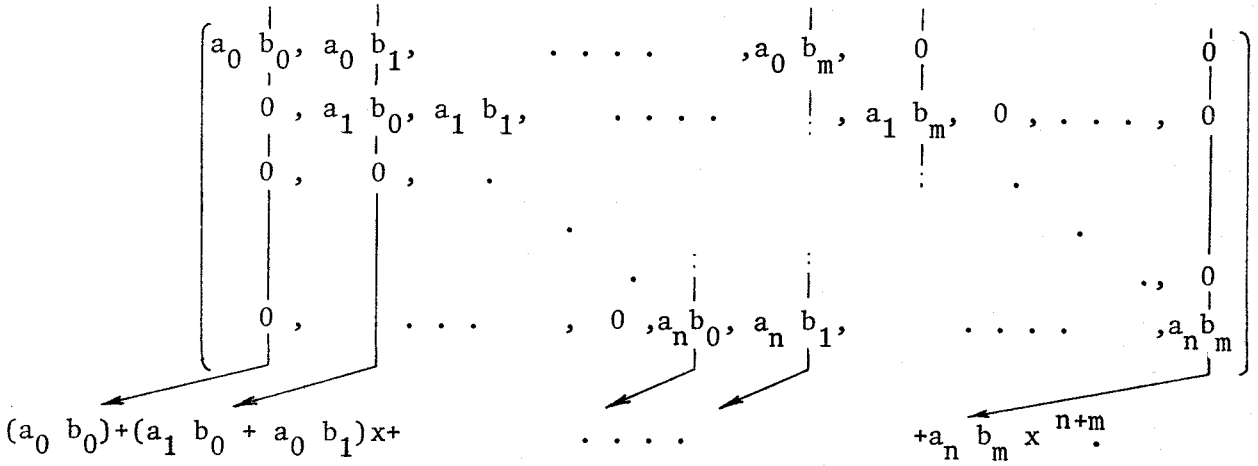

Figure 2. Formation of polynomial product by rotation of outer product. The padded outer product is skewed along its last axis, and then reduced by summation along its first axis.

DOCUMENT CONTROL DATA SHEET

Security classification of this page: UNCLASSIFIED

1 DOCUMENT NUMBERS	
AR Number:	AR-001-179
Report Number:	ERL-0004-TR
Other Numbers:	

2 SECURITY CLASSIFICATION	
a. Complete Document:	UNCLASSIFIED
b. Title in Isolation:	UNCLASSIFIED
c. Summary in Isolation:	UNCLASSIFIED

3 TITLE

POLYNOMIAL MANIPULATION WITH APL

4 PERSONAL AUTHOR(S):

B. Billard

5 DOCUMENT DATE: April 1978

6 6.1 TOTAL NUMBER OF PAGES: 20

6.2 NUMBER OF REFERENCES: 12

7 7.1 CORPORATE AUTHOR(S):

Electronics Research Laboratory

7.2 DOCUMENT SERIES AND NUMBER

Electronics Research Laboratory 0004-TR

8 REFERENCE NUMBERS

a. Task:

b. Sponsoring Agency:

9 COST CODE: 228753/135

10 IMPRINT (Publishing organisation)

Defence Research Centre Salisbury

11 COMPUTER PROGRAM(S) (Title(s) and language(s))

12 RELEASE LIMITATIONS (of the document):

Approved for public release.

12.0	OVERSEAS	NO		P.R.	1	A		B		C		D		E	

Security classification of this page: UNCLASSIFIED

Security classification of this page: UNCLASSIFIED

13 ANNOUNCEMENT LIMITATIONS (of the information on these pages):

No limitation

14 DESCRIPTORS:		15 COSATI CODES:
a. EJC Thesaurus Terms	Polynomials Computer systems programs Symbolic programming Electronic computers Computer programming	0902 1201
b. Non-Thesaurus Terms	APL (programming language) Symbolic computing	

16 LIBRARY LOCATION CODES (for libraries listed in the distribution):

SW SR SD AACA

17 SUMMARY OR ABSTRACT:
(if this is security classified, the announcement of this report will be similarly classified)

A simple but effective system for the manipulation of polynomials of several variables in APL is presented. The system is especially advantageous in situations where more sophisticated symbolic computing systems are not available, or have failed to solve particular problems. The system is shown to successfully solve a problem not resolved by a more sophisticated system.

Security classification of this page: UNCLASSIFIED